MapReduce

2020年8月22日 14:29

-
- MapReduce和YARN

1. MapReduce
   a. 最先是谷歌提出的分布式并行编程模型，Hadoop提供了开源实现
   b.

	传统并行计算框架	MapReduce
集群架构/容错性	共享式(共享内存/共享存储)，容错性差	非共享式，容错性好
硬件/价格/扩展性	刀片服务器、高速网、SAN，价格贵，扩展性差	普通PC机，便宜，扩展性好
编程/学习难度	what-how，难	what，简单
适用场景	实时、细粒度计算、计算密集型	批处理、非实时、数据密集型

   c. 采用“分而治之”策略，一个存储在分布式文件系统中的大规模数据集，会被切分成许多独立的分片（split），这些分片可以被多个Map任务并行处理
   d. MapReduce设计的一个理念就是“计算向数据靠拢”，而不是“数据向计算靠拢”，因为，移动数据需要大量的网络传输开销
   e. MapReduce框架采用了Master/Slave架构，包括一个Master和若干个Slave。Master上运行JobTracker，Slave上运行TaskTracker
   f. Hadoop框架是用Java实现的，但是，MapReduce应用程序则不一定要用Java来写
   g.

函数	输入	输出	说明
Map	<k1,v1> 如: <行号,"a b c">	List(<k2,v2>) 如: <"a",1> <"b",1> <"c",1>	1.将小数据集进一步解析成一批<key,value>对，输入Map函数中进行处理 2.每一个输入的<k1,v1>会输出一批<k2,v2>。<k2,v2>是计算的中间结果
Reduce	<k2,List(v2)> 如：<"a",<1,1,1>>	<k3,v3>如: <"a",3>	输入的中间结果<k2,List(v2)>中的List(v2)表示是一批属于同一个k2的value

2. MapReduce体系结构
   a. Client
      i. 用户编写的MapReduce程序通过Client提交到JobTracker端
      ii. 用户可通过Client提供的一些接口查看作业运行状态
   b. TaskTracker
      i. TaskTracker 会周期性地通过“心跳”将本节点上资源的使用情况和任务的

运行进度汇报给JobTracker，同时接收JobTracker 发送过来的命令并执行相应的操作（如启动新任务、杀死任务等）

ii. TaskTracker 使用“slot”等量划分本节点上的资源量（CPU、内存等）。一个Task 获取到一个slot 后才有机会运行，而Hadoop调度器的作用就是将各个TaskTracker上的空闲slot分配给Task使用。slot 分为Map slot 和Reduce slot 两种，分别供MapTask 和Reduce Task 使用

c. Task

i. Task 分为Map Task 和Reduce Task 两种，均由TaskTracker 启动

d. JobTracker

i. JobTracker负责资源监控和作业调度

ii. JobTracker监控所有TaskTracker与Job的健康状况，一旦发现失败，就将相应的任务转移到其他节点

iii. JobTracker会跟踪任务的执行进度、资源使用量等信息，并将这些信息告诉任务调度器（TaskScheduler），而调度器会在资源出现空闲时，选择合适的任务去使用这些资源

e. mapreduce1.0的缺陷

i. 存在单点故障

ii. JobTracker“大包大揽”导致任务过重（任务多时内存开销大，上限4000节点）

iii. 容易出现内存溢出（分配资源只考虑MapReduce任务数，不考虑CPU、内存）

iv. 资源划分不合理（强制划分为slot，包括Map slot和Reduce slot）

f. hadoop2.0的改进思路

i. 将mapreduce的资源管理调度功能分离出来，形成YARN使得mapreduce称为纯计算框架

ii. Jobtracker的功能分给ResourceManager和ApplicationMaster

iii. TaskTracker的功能分给NodeManager

iv. 资源调度器分类

1) 集中式调度器

2) 双层调度器-Yarn，Mesos等

3) 状态共享调度器

3. YARN（Yet Another Resource Nagotiator）体系结构

a. ResourceManager：全局的资源管理器，负责整个系统的资源管理和分配，主要包括两个组件，调度器（Scheduler）和应用程序管理器（Applications Manager）

i. 主要工作有：

1) 处理客户端请求

2) 启动/监控ApplicationMaster

3) 监控NodeManager

4) 资源分配与调度

ii. 调度器接收来自ApplicationMaster的应用程序资源请求，把集群中的资源以“容器”的形式分配给提出申请的应用程序，容器的选择通常会考虑应用程序所要处理的数据的位置，进行就近选择，从而实现“计算向数据靠拢”
iii. 容器（Container）作为动态资源分配单位，每个容器中都封装了一定数量的CPU、内存、磁盘等资源，从而限定每个应用程序可以使用的资源量
iv. 调度器被设计成是一个可插拔的组件，YARN不仅自身提供了许多种直接可用的调度器，也允许用户根据自己的需求重新设计调度器
v. 应用程序管理器（Applications Manager）负责系统中所有应用程序的管理工作，主要包括应用程序提交、与调度器协商资源以启动ApplicationMaster、监控ApplicationMaster运行状态并在失败时重新启动等

b. ApplicationMaster：ResourceManager接收用户提交的作业，按照作业的上下文信息以及从NodeManager收集来的容器状态信息，启动调度过程，为用户作业启动一个ApplicationMaster
i. 主要工作有：
1）为应用程序申请资源，并分配给内部任务
2）任务调度、监控与容错
ii. 当用户作业提交时，ApplicationMaster与ResourceManager协商获取资源，ResourceManager会以容器的形式为ApplicationMaster分配资源
iii. 把获得的资源进一步分配给内部的各个任务（Map任务或Reduce任务），实现资源的“二次分配”
iv. 与NodeManager保持交互通信进行应用程序的启动、运行、监控和停止，监控申请到的资源的使用情况，对所有任务的执行进度和状态进行监控，并在任务发生失败时执行失败恢复（即重新申请资源重启任务）
v. 定时向ResourceManager发送“心跳”消息，报告资源的使用情况和应用的进度信息
vi. 当作业完成时，ApplicationMaster向ResourceManager注销容器，执行周期完成

c. NodeManager
i. 主要工作有：
1）单个节点上的资源管理
2）处理来自ResourceManger的命令
3）处理来自ApplicationMaster的命令
ii. 驻留在一个YARN集群中的每个节点上的代理，主要负责：
1）容器生命周期管理
2）监控每个容器的资源（CPU、内存等）使用情况
3）跟踪节点健康状况
4）以“心跳”的方式与ResourceManager保持通信
5）向ResourceManager汇报作业的资源使用情况和每个容器的运行状态
6）接收来自ApplicationMaster的启动/停止容器的各种请求

iii. 需要说明的是，NodeManager主要负责管理抽象的容器，只处理与容器相关的事情，而不具体负责每个任务（Map任务或Reduce任务）自身状态的管理，因为这些管理工作是由ApplicationMaster完成的，ApplicationMaster会通过不断与NodeManager通信来掌握各个任务的执行状态

d. 在集群部署方面，YARN的各个组件是和Hadoop集群中的其他组件进行统一部署的

4. YARN工作流程
   a. 用户编写客户端应用程序，向YARN提交应用程序，提交的内容包括ApplicationMaster程序、启动ApplicationMaster的命令、用户程序等
   b. YARN中的ResourceManager负责接收和处理来自客户端的请求，为应用程序分配一个容器，在该容器中启动一个ApplicationMaster
   c. ApplicationMaster被创建后会首先向ResourceManager注册
   d. ApplicationMaster采用轮询的方式向ResourceManager申请资源
   e. ResourceManager以“容器”的形式向提出申请的ApplicationMaster分配资源
   f. 在容器中启动任务（运行环境、脚本）
   g. 各个任务向ApplicationMaster汇报自己的状态和进度
   h. 应用程序运行完成后，ApplicationMaster向ResourceManager的应用程序管理器注销并关闭自己
5. YARN与mapreduce1.0对比
   a. 客户端并没有发生变化，其大部分调用API及接口都保持兼容，因此，原来针对Hadoop1.0开发的代码不用做大的改动，就可以直接放到Hadoop2.0平台上运行
   b. 大大减少了承担中心服务功能的ResourceManager的资源消耗
      i. ApplicationMaster来完成需要大量资源消耗的任务调度和监控
      ii. 多个作业对应多个ApplicationMaster，实现了监控分布化
   c. MapReduce1.0既是一个计算框架，又是一个资源管理调度框架，但是，只能支持MapReduce编程模型。而YARN则是一个纯粹的资源调度管理框架，在它上面可以运行包括MapReduce在内的不同类型的计算框架，只要编程实现相应的ApplicationMaster
   d. YARN中的资源管理比MapReduce1.0更加高效（以容器为单位，而不是以slot为单位）
6. YARN的发展目标：一个集群多个框架
   a. 一个企业当中同时存在各种不同的业务应用场景，需要采用不同的计算框架
      i. MapReduce实现离线批处理
      ii. Impala实现实时交互式查询分析
      iii. Storm实现流式数据实时分析
      iv. Spark实现迭代计算
   b. 这些产品通常来自不同的开发团队，具有各自的资源调度管理机制，为了避免不同类型应用之间互相干扰，企业就需要把内部的服务器拆分成多个集群，分别安装运行不同的计算框架，即“一个框架一个集群”，导致问题

i. 集群资源利用率低
ii. 数据无法共享
iii. 维护代价高

c. YARN的目标就是实现“一个集群多个框架”，即在一个集群上部署一个统一的资源调度管理框架YARN，在YARN之上可以部署其他各种计算框架
i. 由YARN为这些计算框架提供统一的资源调度管理服务，并且能够根据各种计算框架的负载需求，调整各自占用的资源，实现集群资源共享和资源弹性收缩
ii. 可以实现一个集群上的不同应用负载混搭，有效提高了集群的利用率
iii. 不同计算框架可以共享底层存储，避免了数据集跨集群移动

7. MapReduce工作流程
a. 概述
i. 不同的Map任务之间不会进行通信
ii. 不同的Reduce任务之间也不会发生任何信息交换
iii. 用户不能显式地从一台机器向另一台机器发送消息
iv. 所有的数据交换都是通过MapReduce框架自身去实现的

b. split分片
i. HDFS 以固定大小的block 为基本单位存储数据，而对于MapReduce 而言，其处理单位是split。split 是一个逻辑概念，它只包含一些元数据信息，比如数据起始位置、数据长度、数据所在节点等。它的划分方法完全由用户自己决定
ii. Hadoop为每个split创建一个Map任务，split 的多少决定了Map任务的数目。大多数情况下，理想的分片大小是一个HDFS块
iii. 最优的Reduce任务个数取决于集群中可用的reduce任务槽(slot)的数目，通常设置比reduce任务槽数目稍微小一些的Reduce任务个数（这样可以预留一些系统资源处理可能发生的错误）

c. shuffle
i. 溢写：每个Map任务分配一个缓存（MapReduce默认100MB缓存）、设置溢写比例0.8，分区默认采用哈希函数，排序是默认的操作，排序后可以合并（Combine），合并不能改变最终结果
ii. 在Map任务全部结束之前进行归并，归并得到一个大的文件，放在本地磁盘，文件归并时，如果溢写文件数量大于预定值（默认是3）则可以再次启动Combiner，少于3不需要，JobTracker会一直监测Map任务的执行，并通知Reduce任务来领取数据
iii. Reduce任务通过RPC向JobTracker询问Map任务是否已经完成，若完成，则领取数据，Reduce领取数据先放入缓存，来自不同Map机器，先归并，再合并，写入磁盘，多个溢写文件归并成一个或多个大文件，文件中的键值对是排序的,当数据很少时，不需要溢写到磁盘，直接在缓存中归并，然后输出给Reduce
iv. 注：<a,1>和<a,1>combine生成<a,2>而merge则会生成<a,<1,1>>

d. 局限
   i. 性能慢—文件IO和网络传输，可采用Spark
   ii. 过于底层，Map和reduce两个函数的编写麻烦，可采用其他工具，如：对于数据库查询的Mapreduce操作用HIVE
   iii. 并不是所有算法都能mapreduce，如很多机器学习算法

8. mapReduce实现（以单词计数为例，注意数据要可序列化可网络IO）
   a. 编写Map处理逻辑（泛型参数为输入输出的key和value，object是行号），例

```
public class MyMapper extends Mapper<Object, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

   b. 编写Reduce处理逻辑，例

```
public class MyReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOEx
ception, InterruptedException {
        int sum = 0;
        for (IntWritable val : values)
            sum += val.get();
        result.set(sum);
        context.write(key, result);
    }
}
```

   c. 编写main方法，例

```
public class WordCount {
    public static void main(String[] args) throws IOException, ClassNotFoundException, Inte
rruptedException {
        Configuration conf = new Configuration();  //程序运行时参数
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs(); //获
取命令行输入参数
        if (otherArgs.length != 2) {
            System.err.println("Usage: wordcount <in> <out>");
            System.exit(2);
        }
        Job job = Job.getInstance(conf,"word count");  //设置环境参数
        job.setJarByClass(WordCount.class);  //设置整个程序的类名
        job.setMapperClass(MyMapper.class);  //添加MyMapper类
        job.setReducerClass(MyReducer.class);  //添加MyReducer类
        job.setOutputKeyClass(Text.class);  //设置输出类型
        job.setOutputValueClass(IntWritable.class);  //设置输出类型
        // job.setCombinerClass(IntSumReducer.class);//开启combine
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  //设置输入文件
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1])); //设置输出文件
        System.exit(job.waitForCompletion(true)?0:1);//提交任务到hadoop
    }
}
```

   d. 编译打包代码以及运行程序
      i. 使用java编译程序，打包为jar包
      ii. 上传JAR包到hadoop节点，数据文件到hdfs
      iii. hadoop jar xxx.jar 输入数据文件的hdfs目录 输出文件的hdfs目录
         1) （需要启动Hadoop，运行yarn和hdfs）
         2) 输出文件不能事先写好，要让hadoop自行创建

iv. 查看结果

e. Hadoop中执行MapReduce任务的几种方式
   i. Hadoop jar
   ii. Pig
   iii. Hive
   iv. Python
   v. Shell脚本
   vi. 在解决问题的过程中，开发效率、执行效率都是要考虑的因素，不要太局限于某一种方法
f. 编写容易，优化难，调优是关键